This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

Reliable Virtual Machine Placement and Routing in Clouds

Song Yang, Philipp Wieder, Ramin Yahyapour, Stojan Trajanovski, Xiaoming Fu, *Senior Member, IEEE*

Abstract—In current cloud computing systems, when leveraging virtualization technology, the customer's requested data computing or storing service is accommodated by a set of communicated virtual machines (VM) in a scalable and elastic manner. These VMs are placed in one or more server nodes according to the node capacities or failure probabilities. The VM placement availability refers to the probability that at least one set of all customer's requested VMs operates during the requested lifetime. In this paper, we first study the problem of placing at most H groups of k requested VMs on a minimum number of nodes, such that the VM placement availability is no less than δ, and that the specified communication delay and connection availability for each VM pair under the same placement group are not violated. We consider this problem with and without Shared-Risk Node Group (SRNG) failures, and prove this problem is NP-hard in both cases. We subsequently propose an exact Integer Nonlinear Program (INLP) and an efficient heuristic to solve this problem. We conduct simulations to compare the proposed algorithms with two existing heuristics in terms of performance. Finally, we study the related reliable routing problem of establishing a connection over at most w link-disjoint paths from a source to a destination, such that the connection availability requirement is satisfied and each path delay is no more than a given value. We devise an exact algorithm and two heuristics to solve this NP-hard problem, and evaluate them via simulations.

Index Terms—Virtual machine placement, routing, availability, reliability, cloud computing, optimization algorithms.

1 Introduction

Cloud computing [2] is a distributed computing and storing paradigm, which can provide scalable and reliable service over the Internet for on-demand data-intensive applications (e.g., on-line search or video streaming) and data-intensive computing (e.g., analyzing and processing a large volume of scientific data). The key features of cloud computing, including "pay-as-you-go" and "elastic service", attract many service providers and customers to deploy their workload from their own infrastructures or platforms to public or private clouds.

Distributed cloud systems are usually composed of distributed inter-connected data centers, which leverage virtualization technology to provide computing and storage service for each on-demand request. Once a request arrives, several virtual machines (VM) are created in one or more server nodes (which may be located in the same or different data centers) in order to accommodate the request. However, the server node failures caused by hardware malfunctions such as hard disk or memory module failures and software problems such as software bugs or configuration errors may result in the loss of the VMs hosted on it and hence the whole service cannot be guaranteed. An efficient way to overcome this concern is to create and place more VM replicas, but this approach should also take the nodes' availabilities into account. For instance, if all the VMs together with their replicas are placed at nodes with high failure probability, then a proper service cannot be guaranteed. The VM placement availability, a value between 0 and 1, is therefore important and refers to the probability that at least one set of all customer's requested VMs is in the operating state during the entire requested lifetime.

Moreover, if two or more VMs are placed on different nodes, we should also ensure reliable communications between these VMs. In fact, a single unprotected path will fail if one of the links belonging to it fails. To increase the reliability of transporting data from a source to a destination, path protection (or survivability) is called for. For instance, by allocating a pair of link-disjoint paths from a source to a destination, the data is transported on the primary path. Upon a failure of the primary path, the data can be switched to the backup path. However, the path protection mechanism, which does not allow for more than 2 link-disjoint paths, may still be not reliable enough and $w > 2$ link-disjoint paths may be needed. Moreover, the link availability should also be taken into account. For a connection over at most w link-disjoint paths between a node pair, its availability specifies the probability that at least one path is operational. Connection availability is therefore important to quantitatively measure the availability of delivering data between VMs located on different nodes in a cloud.

In this paper, we first study the Reliable VM Placement (RVMP) problem, which is to place at most H groups of k requested VMs on a minimum number of nodes, such that the VM placement availability is no less than δ, and the specified communication delay and connection availability for each VM pair are not violated.

Following that, we study the Availability-Based Delay-

- *S. Yang and P. Wieder are with Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany. E-mail: {S.Yang, P.Wieder}@gwdg.de*
- *R. Yahyapour is with GWDG and Institute of Computer Science, University of Göttingen, Göttingen, Germany. E-mail: R.Yahyaour@gwdg.de*
- *This work was done while S. Trajanovski was with the University of Amsterdam and Delft University of Technology, The Netherlands. S. T. is now with Philips Research and Delft University of Technology. E-mail: S.Trajanovski@tudelft.nl*
- *X. Fu is with Institute of Computer Science, University of Göttingen, Göttingen, Germany. E-mail: Fu@cs.uni-goettingen.de*

A preliminary part of this paper appeared as conference publication [1].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

Constrained Routing (ABDCR) problem, which is to establish a connection over at most w (partially) link-disjoint paths from a source to a destination such that the connection availability is at least η and each path has a delay no more than D. Our key contributions are as follows:

- We propose a mathematical model to formulate VM placement availability with and without Shared-Risk Node Group failures, and prove that the Reliable VM Placement (RVMP) problem under both cases is NP-hard.
- We propose an Integer Nonlinear Program (INLP) and a heuristic to solve the RVMP problem.
- We compare the proposed algorithms with two existing heuristics in terms of performance via simulations.
- We prove that the ABDCR problem is NP-hard, devise an exact algorithm and two heuristics to solve it, and further verify them.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 and 4 formulate the VM placement availability calculation without and with SRNG failures, respectively. In Section 5, we study the Reliable VM Placement (RVMP) problem and prove it is NP-hard. We propose an exact Integer Nonlinear Program (INLP) and a heuristic to solve the RVMP problem. The proposed algorithms are also evaluated via simulations. In Section 6, we define the Availability-Based Delay-Constrained Routing (ABDCR) problem, prove the problem is NP-hard, and propose an exact algorithm and two heuristics to solve it. We also conduct simulations to verify the proposed algorithms as well. Finally, we conclude in Section 7.

2 Related Work

A high-level comprehensive survey about VM placement can be found in [3] [4].

2.1 Network-Aware VM Placement

Alicherry and Lakshman [5] first investigate how to place requested VMs on distributed data center nodes such that the maximum length (e.g., delay) of placed VM pairs is minimized. A 2-approximation algorithm is proposed to solve this problem when a triangle link length is assumed. They subsequently study how to place VMs on physical machines (racks and servers) within a data center in order to minimize the total inter-rack communication costs. Assuming that the topology of the data center is a tree, they devise an exact algorithm to solve this problem. Finally, they propose a heuristic for partitioning VMs into disjoint sets (e.g., racks) such that the total communication costs between VMs belonging to different partitions is minimized.

Biran *et al.* [6] address the VM placement problem by minimizing the min-cut ratio in the network, which is defined as the used capacity of the cut links consumed by the communication of VMs divided by the total capacity of the cut links. They prove this problem is NP-hard and propose two efficient heuristics to solve it. Jiang *et al.* [7] jointly consider the VM placement and routing problem within one data center network. They propose an approximation on-line algorithm leveraging the technique of Markov approximation.

Meng *et al.* [8] address the problem of assigning VMs to slots (CPU/memory on a host) within a data center network in order to minimize total network costs. They prove the problem is NP-hard and propose a heuristic that tries to assign VMs with large mutual rate requirement close to each other.

2.2 Reliable VM Placement

Israel and Raz [9] study the Virtual Machine Recovery Problem (VMRP). The VMRP is to place the backup VMs for their corresponding servicing VMs on either active or inactive host, which needs to strike a balance between the (active) machine maintenance cost and VM recovery Service Level Agreement (e.g., recovery time). They show that the VMRP is NP-hard, and they propose a bicriteria approximation algorithm and an efficient heuristic to solve it.

Bin *et al.* [10] tackle the VM placement problem by considering k-resiliency constraint to guarantee high availability goals. A VM is marked as k-resilient, if its current host fails and there are up to $k-1$ additional host failures, and it can still be guaranteed to relocate to a non-failed host. In this sense, a placement is said to be k-resilient if it satisfies the k-resiliency requirements of all its VMs. They first formulate this problem as a second order optimization statement and then transform it to a generic constraint program in polynomial time.

Zhu *et al.* [11] address the Reliable Resource Allocation (RRA) problem. In this problem, each node has a capacity limit of storing VMs and each link is associated with an availability value (between 0 and 1). The problem is to find a star of a network to place the requested VMs, such that the node capacity limit is obeyed and the availability of the star is no less than the specified. They prove that the RRA problem is NP-hard and propose an exact algorithm as well as a heuristic to solve it. However, the defined problem in [11] does not consider the node's availability and also it restricts to find a star instead of an arbitrary subgraph.

Li and Qian [12] assume that the VM reliability requirement is equal to the maximum fraction of VMs of the same function that can be placed in a rack. Yang *et al.* [13] develop a variance-based metric to measure the risk of violating the VM placement availability requirement, but none of them take VM replicas/backups into account. Nevertheless, none of above papers quantitatively model the availability of VM placement (and solve the respective reliable VM placement problem), as we do in this paper.

2.3 Availability-Aware Routing

Song *et al.* [14] propose an availability-guaranteed routing algorithm, where different protection types are allowed. They define a new cost function for computing a backup path when the unprotected path fails to satisfy the availability requirement. She *et al.* [15] prove the problem of finding two link-disjoint paths with maximal reliability (availability) is NP-hard. They also propose two heuristics for that problem. Luo *et al*. [16] address the problem of finding one unprotected path or a pair of link-disjoint paths, such that the cost of the entire path(s) is minimized and the reliability

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

requirement is satisfied. To solve it, they propose an exact ILP as well as two approximation algorithms. However, the reliability (availability) calculation in [16] is different from the aforementioned papers, and assumes a single-link failure model. Assuming each link in the network has a failure probability (=1-availability), Lee *et al.* [17] minimize the total failure probability of unprotected, partially link-disjoint and fully link-disjoint paths by establishing INLPs. They further transform the proposed INLPs to ILPs by using linear approximations. Yang *et al.* [18], [19] study the availability-based path selection problem, which is to find at most w (partially) link-disjoint paths and for which the total availability is no less than the specified. They prove that this problem is NP-hard and cannot be approximated to an arbitrary degree when $w \geq 2$. They propose an exact INLP and a heuristic to solve this problem.

3 VM Placement Availability

The availability of a system is the fraction of time that the system is operational during the entire service time. The availability A_j of a network component j can be calculated as [20]:

$$A_j = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

where $MTTF$ represents Mean Time To Failure and $MTTR$ denotes Mean Time To Repair. In this paper, a node in the network represents a server, and its availability is equal to the product of the availabilities of all its components (e.g., hard disk, memory, etc.). In reality, we can obtain the server's availability value by accessing the detailed logs extracting every hardware component repair/failure incident during the lifetime of the server. The details for characterizing server and other data center network device (e.g., switches) failures can be found in [21] and [22]. Since our focus in this paper is not on how to calculate the device's availability, we assume that the server availabilities (or the SRNG event failure probabilities) value are known. Moreover, we assume a general multiple node (link) failure scenario, which means at one particular time point, multiple nodes (links) may fail. In this section, we first assume that the node availabilities are uncorrelated/independent.

We assume that the user request consists of k VMs with associated communication requirements (we consider delay and connection availability in this paper) between different VM pairs. These k VMs are represented by $v_1, v_2, \ldots, v_k$. For each requested VM v_i ($1 \leq i \leq k$), placing it on the same node (say n) more than once cannot increase placement availability, since when n fails, all its resident VMs will fail simultaneously. Therefore, we need to place v_i on different nodes to increase the placement availability. Let us use H_i to represent the maximum number of nodes to host VM v_i. Or, equivalently, H_i indicates the maximum number of nodes that v_i can be placed on. We denote $H = \max_{i=1}^{k}(H_i)$. We distinguish and analyze the VM placement availability under two different cases, namely (1) Single Placement: each VM is placed on exactly $H = 1$ node in the network, and (2) Protected Placement: $\exists v_j \in V$, such that v_j can be placed on $H_j > 1$ nodes in the network, i.e., $H > 1$. Without loss of generality, in this paper, we assume that the servers are heterogeneous and they can be located in either the same data center or different data centers.

In the single placement case, if m nodes with availability $A_1, A_2, \ldots, A_m$ are used for hosting k VMs ($m \leq k$), then the availability (denoted by A_p) of this VM placement is:

$$A_p = A_1 \cdot A_2 \cdots A_m \tag{2}$$

Eq. (2) indicates that since k VMs are requested in total, the availability should take into account the probability that all these k VMs are operational.

In the protected placement case, there exist one or more VMs that can be placed on at most H nodes. Therefore, we regard that a protected placement P is composed of (maximum) H single placements. Within each single placement, the communication requirements between VM pairs should be satisfied. For the ease of clarification, we further term each of the H single placements in the protected placement as placement group p_i, which means the "$i-$th" placing k VMs on m_i nodes, where $1 \leq i \leq H$ and $1 \leq m_i \leq k$. We regard p_1 as the primary placement group. We make no difference between the single placement and the placement group. Since different placement groups may place one or more VMs on the same node, we distinguish the protected placement as two cases, namely (1) *fully protected placement*, for each VM $v \in V$, v is placed by each group p_i ($1 \leq i \leq H$) at H different nodes, and (2) *partially protected placement*, $\exists v \in V$, such that v is placed on less than H nodes, i.e., two or more placement groups place v on the same node.

In the fully protected placement case, the availability can be calculated as:

$$A_{PD}^{F} = 1 - \prod_{i=1}^{H}(1 - A_{p_i}) = \sum_{i=1}^{H} A_{p_i} - \sum_{0<i<j\leq H} A_{p_i} \cdot A_{p_j} + \sum_{0<i<j<u\leq H} A_{p_i} \cdot A_{p_j} \cdot A_{p_u} + \cdots + (-1)^{H-1} \prod_{i=1}^{H} A_{p_i} \tag{3}$$

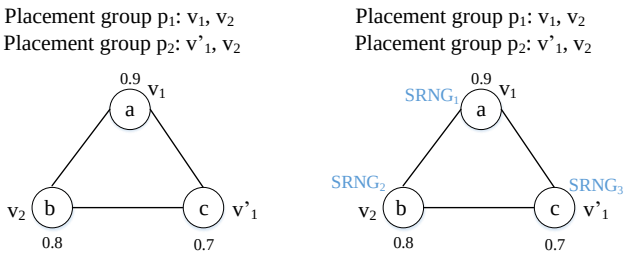
where $A_{p_i} = \prod_{n\in m_i} A_n$ denotes the availability of a single VM placement according to Eq. (2). Eq. (3) reflects that the availability of H placement groups is equivalent to the probability that at least one single placement (a set of k VMs) is operational in the service-life time.

In the partially protected placement case, if one VM is placed on less than H nodes, we could regard that this VM is jointly placed by more than one placement group. For example, in Fig. 1(a), each node is associated with its own availability value and we need to place two VMs (v_1 and v_2) on it. We set $H_1 = 2$ and $H_2 = 1$ for simplicity. We assume that placement group p_1 places v_1 on node a, and placement group p_2 places v_1's replica (denoted by v_1') on node c. On the other hand, v_2 is only placed on one node. Therefore, we can regard that p_1 and p_2 jointly place v_2 on node b.

However, we cannot directly apply Eq. (3) to calculate its availability, since the availabilities of nodes which hold "shared" VMs will be counted more than once. To amend this, we use a new operator $\circ$[1]. Suppose there are m different nodes $n_1, n_2, \ldots, n_m$ with availabilities $A_1, A_2, \ldots, A_m$. For a node n_x with availability A_x, $\circ$ can be defined as follows:

1. As in [18], [19] for the partially link-disjoint paths connection availability.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems



(a) Without SRNG (b) With SRNG

Fig. 1: Partially VM placement availability calculation.

$$A_1 \cdot A_2 \cdots A_m \circ A_x = \begin{cases} \prod_{i=1}^{m} A_i & \text{if } \exists n_i = n_x \\ \prod_{i=1}^{m} A_i \cdot A_x & \text{otherwise} \end{cases} \quad (4)$$

Let $\coprod$ denote consecutive $\circ$ operations of the different sets, then the availability (represented by A_{PD}^{H}) of H partially placement groups can now be represented as:

$$\begin{aligned} A_{PD}^{H} &= 1 - \coprod_{i=1}^{H}(1 - A_{p_i}) \\ &= 1 - (1 - A_{p_1}) \circ (1 - A_{p_2}) \circ \circ \circ (1 - A_{p_H}) \\ &= \sum_{i=1}^{H} A_{p_i} - \sum_{0<i<j\leq H} A_{p_i} \circ A_{p_j} + \\ &\sum_{0<i<j<u\leq H} A_{p_i} \circ A_{p_j} \circ A_{p_u} + \cdots + (-1)^{H-1} \coprod_{i=1}^{H} A_{p_i} \end{aligned} \quad (5)$$

where A_{p_i} denotes the availability of placement group p_i and can be calculated from Eq. (2). Now, going back to the example of Fig. 1(a), when there are no communication requirements between the two requested VMs, the placement availability of p_1 and p_2 is equal to $1-(1-A_a \circ A_b) \circ (1-A_c \circ A_b) = A_a \circ A_b + A_c \circ A_b - A_a \circ \underline{A_b} \circ A_c \circ \underline{A_b} = A_a \cdot A_b + A_c \cdot A_b - A_a \cdot \underline{A_b} \cdot A_c = 0.9 \cdot 0.8 + 0.7 \cdot \overline{0.8} - 0.9 \cdot \overline{0.8} \cdot 0.7 = 0.776$.

In order to emphasize the importance of communication requirements between the VMs, we consider the example of Fig. 2. For simplicity, it is assumed that each node can host at most one VM and its availability value is depicted in Fig. 2. Moreover, we impose that node pairs (a, d) and (b, c) have communication delays bigger than the requested delay, i.e., these two node pairs do not satisfy the VM communication requirement.

As is shown in Fig. 2, there are in total 4 possible placement groups: (v_1, v_2, v_3), (v_1, v_2, v'_3), (v_1, v'_2, v_3) and (v_1, v'_2, v'_3). However, neither (v_1, v'_2, v_3) nor (v_1, v_2, v'_3) can form a feasible placement group, because the communication delay is violated in either of these groups, and therefore, their availability cannot be taken into account.[2] As a result, we only take into account the two placement groups (v_1, v_2, v_3) and (v_1, v'_2, v'_3). According to Eq. (5), their overall availability is $1 - (1 - A_s \circ A_a \circ A_b) \circ (1 - A_s \circ A_c \circ A_d) = A_s \circ A_a \circ A_b + A_s \circ A_c \circ A_c - \underline{A_s} \circ A_a \circ A_b \circ \underline{A_s} \circ A_c \circ A_d = A_s \cdot A_a \cdot A_b + A_s \cdot A_c \cdot A_c - \underline{A_s} \cdot A_a \cdot A_b \cdot A_c \cdot A_d = 0.95 \cdot 0.9 \cdot 0.9 + 0.95 \cdot 0.9 \cdot 0.8 - \overline{0.95} \cdot 0.9 \cdot 0.9 \cdot 0.8 = 0.85614$.

2. As a side note, neglecting the communication requirement of VM pairs, the overall availability of the 4 placement groups would have been: $1 - (1 - A_s \circ A_a \circ A_b) \circ (1 - A_s \circ A_a \circ A_d) \circ (1 - A_s \circ A_c \circ A_b) \circ (1 - A_s \circ A_c \circ A_d) = 0.89376$.

Node pairs (a, d) and (b, c) violate the communication requirement.

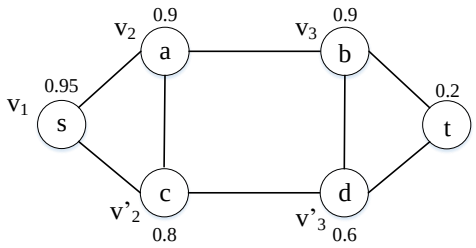


Fig. 2: An example of a partially VM placement availability calculation under communication requirement constraint.

4 Shared-Risk Node Group

In this section, we assume two types of failures/availabilities, namely Shared-Risk Node Group (SRNG) failures and single node failures/availabilities. A SRNG failure [23] reflects that a particular group of nodes will fail simultaneously, i.e., they have correlated failures. For example, in data center networks [24], servers are hosted by different racks, and within the same rack, servers are connected with a Top of Rack (ToR) switch. These ToR switches are further inter-connected through aggregate switches in a tree-like topology. Similarly, these aggregate switches are further connected with several core switches in the upper layer. In this context, the servers which are hosted by the same rack will fail simultaneously, if the rack they belong to fails. Similarly, the failure of a ToR switch will cause the failures of all the racks which are connected with it, that will subsequently cause failures of the hosted servers. One rack failure then corresponds to one distinct SRNG event. One (server) node can belong to multiple SRNG events (e.g., rack failure or switch failure). We assume there are in total g SRNG events, and the failure probability of SRNG event i is represented by π_i. For the ease of elaboration, we let $\lambda_i = 1 - \pi_i$, which can represent the non-occurring probability of SRNG event i. For each node $n \in \mathcal{N}$, we denote R_n as the set of all the SRNG events it belongs to. The VM placement availability should incorporate the SRNG non-occurring probabilities as well as the node availabilities. As a result, if m nodes with availability A_1, A_2,..., A_m are used for hosting k VMs ($m \leq k$) by a single placement p, then the availability of this single placement p can be calculated as follows:

$$\prod_{i;srng_i \cap p \neq \emptyset} \lambda_i \cdot \prod_{j=1}^{m} A_j \quad (6)$$

Similarly, the protected VM placement (fully and partially) availability can be calculated by substituting Eq. (6) with A_{p_i} in Eq. (5). It is worthwhile to mention that the operator $\circ$ in Eq. (5) still holds for λ. For example, in Fig. 1(b), each node is assigned with a SRNG event and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

node availability value. Suppose $\lambda_1 = 0.999$, $\lambda_2 = 0.99$ and $\lambda_3 = 0.9$. Assume that placement p_1 places v_1 on A and v_2 on B, and placement p_2 places v_1 on C and v_2 on B. In this sense, both the SRNG event failure and node availability should only be counted once, especially for $SRNG_2$ and node B (since they both belong to p_1 and p_2). Consequently, the VM placement availability of p_1 and p_2 is:

$$\begin{aligned}
&1-(1-\lambda_1\circ\lambda_2\circ A_a\circ A_b)\circ(1-\lambda_2\circ\lambda_3\circ A_b\circ A_c)\\
=&\lambda_1\circ\lambda_2\circ A_a\circ A_b+\lambda_2\circ\lambda_3\circ A_b\circ A_c-\\
&\lambda_1\circ\underline{\lambda_2}\circ A_a\circ\underline{A_b}\circ\underline{\lambda_2}\circ\lambda_3\circ\underline{A_b}\circ A_c\\
=&\lambda_1\cdot\lambda_2\cdot A_a\cdot A_b+\lambda_2\cdot\lambda_3\cdot A_b\cdot A_c-\lambda_1\cdot\underline{\lambda_2}\cdot A_a\cdot\underline{A_b}\cdot\lambda_3\cdot A_c\\
=&0.999\cdot0.99\cdot0.9\cdot0.8+0.99\cdot0.9\cdot0.8\cdot0.7-\\
&0.999\cdot0.99\cdot0.9\cdot0.8\cdot0.9\cdot0.7=0.762432264
\end{aligned}$$

5 Reliable Virtual Machine Placement

5.1 Problem Definition

We denote by $\mathcal{N}$ the set of N server nodes and by $\mathcal{L}$ the set of L links between them. The server nodes in $\mathcal{N}$ form a complete graph[3] ($L = \frac{N(N-1)}{2}$). Each node $N_j \in \mathcal{N}$ has a storage upper bound of s_j. For each link $(m,n) \in \mathcal{L}$, a function $F(m,n,\eta,D)$ returns 1 if m and n can have a connection availability of at least η and communication delay of at most D, and 0 otherwise. A request is denoted by $r(k,c,V,T,A,\delta)$, where k indicates the requested number of VMs V with demanding capacity c_v ($v \in V$). T and A are $k \times k$ matrices, which specify the delay constraint and connection availability constraint between any two VMs, respectively, and δ is the requested VM placement availability.

Formally, the Reliable VM Placement (RVMP) problem is defined as follows:

Definition 1. For a request $r(k,c,V,T,A,\delta)$, the Reliable VM Placement (RVMP) problem is to place at most H groups of k VMs on a minimum number of nodes such that:

- The VM placement availability is no less than δ.
- Each node does not exceed its storage limit.
- Any two VMs i_1 and i_2 under the same placement group have a communication delay no more than $T(i_1,i_2)$, and a connection availability no less than $A(i_1,i_2)$.

In the RVMP problem, we assume that each VM can be placed at up to H different nodes. Moreover, we only consider the server nodes and ignore some other nodes in the network (e.g., router nodes, switch nodes). In fact, the link between each node pair in the RVMP problem actually implies a (set of) path(s) which may traverse some other intermediate nodes. Finding reliable and delay-sensitive paths could be easier within a tree-like data center network, but this problem becomes harder when the node pairs are located in different data centers (a more general network). As we proved in [18], [19], the problem of finding $w \geq 2$ link-disjoint paths for which the connection availability is no less than a given value is already NP-hard and cannot be approximated to an arbitrary degree. In this sense, jointly considering the RVMP problem and reliable routing problem will make it even harder to solve. Therefore, we assume $F(m,n,\eta,D)$ is precalculated by using the algorithms proposed in Section 6, where we will address how to find reliable and delay-sensitive paths by taking link availability and link delay into account.

Theorem 1. The RVMP problem is NP-hard.

Proof: Let us first introduce the NP-hard Bin-Packing problem [25]: Given n items with sizes e_1, e_2, ..., e_n, and a set of m bins with capacity c_1, c_2, ..., c_m, the Bin-Packing problem is to pack all the items into minimized number of bins without violating the bin capacity size. If we assume that for each node pair (m,n), $F(m,n,\eta,D) = 1$ for any η and D and all the nodes have availability 1, then the RVMP problem for $H = 1$ is equivalent to the Bin-Packing problem, which is NP-hard. Next, let us analyze its complexity when the objective of minimizing the number of used nodes is not considered.

- Each node has unlimited storage: In this case, each set of k VMs can be placed on one node and we need to find H nodes in the network to store each set of k VMs. This can be solved in $\binom{N}{H}$ searching when $N > H$ or using N nodes to host N groups of k VMs when $N \leq H$, which is polynomial time solvable.
- Each node has limited storage: Assume $H = 1$ and $A_n = 1$, $\forall n \in \mathcal{N}$. Moreover, assume a certain D value and that $(m,n) \in \mathcal{L}$, $F(m,n,\eta,D)$ remains the same for any η. That is, the link between each node pair is only assigned with a delay value (connection availability is not taken into account). Under this assumption, Alicherry and Lakshman [26] have proved that the RVMP problem can be reduced to the 3SAT problem, and cannot be approximated to an arbitrary degree.

□

The RVMP problem with SRNG failures is also NP-hard and cannot be approximated to an arbitrary degree, when we assume that each node is associated with one distinct SRNG event and all the node availabilities are assumed to be 1. In the following, we will devise both an exact solution and a heuristic to solve the RVMP problem.

5.2 Exact Solution

In this subsection, we propose an exact Integer Nonlinear Program (INLP) to solve the RVMP problem. We first solve the RVMP problem without SRNG failures and start by explaining the necessary notations and variables:

INLP notations:

$r(k,c,V,T,A,\delta)$: A VM placement request r as specified in Section 5.1.

$\mathcal{N}, \mathcal{L}$: set of N nodes and set of L links, respectively.

H: The maximum number of times for one VM to be placed in the network.

$F(m,n,\eta,D)$ Returns 1 if a connection exists between m and n such that connection availability is at least η and communication delay value is at most D, and 0 otherwise.

λ_i^n: The non-occurring probability of the i-th SRNG if node n belongs to it, and 1 otherwise.

INLP variable:

3. There is a possibility of all nodes to be connected, but the quality/goodness of these connections are determined by other factors (e.g., connection availability, communication delay).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

P_{vn}^h: a binary variable and it is equal to 1 if VM v is placed on node n by placement group h, and 0 otherwise, where $v \in V$, $n \in \mathcal{N}$ and $1 \leq h \leq H$.

Objective:

$$\min \sum_{n \in \mathcal{N}} \left(\max_{1 \leq h \leq H, v \in V} P_{vn}^h \right) \tag{7}$$

Placement constraint:

$$\sum_{n \in \mathcal{N}, 1 \leq h \leq H} P_{vn}^h \geq 1 \quad \forall v \in V \tag{8}$$

Storage constraint:

$$\sum_{v \in V} \left(\max_{h=1}^{H} P_{vn}^h \right) \cdot c_v \leq s_n \quad \forall n \in \mathcal{N} \tag{9}$$

Delay and connection availability constraint:

$$F(m, n, A(m,n), T(m,n)) \cdot P_{am}^h \cdot P_{bn}^h = 1$$
$$\forall 1 \leq h \leq H, (m,n) \in \mathcal{L}, 1 \leq a, b \leq k : a \neq b \tag{10}$$

VM placement availability constraint:

$$\sum_{h=1}^{H} \prod_{n \in \mathcal{N}} \min_{v \in V} \left(1 - P_{vn}^h + P_{vn}^h A_n\right) -$$
$$\sum_{1 \leq h < u \leq H} \prod_{n \in \mathcal{N}} \min \left(\min_{v \in V}(1 - P_{vn}^h + P_{vn}^h A_n), \min_{v \in V}(1 - P_{vn}^u + P_{vn}^u A_n) \right)$$
$$+ \cdots + (-1)^{H-1} \left(\prod_{n \in \mathcal{N}} \min_{1 \leq h \leq H} (\min_{v \in V}(1 - P_{vn}^h + P_{vn}^h A_n)) \right) \geq \delta \tag{11}$$

Eq. (7) minimizes the number of total used nodes. For instance, we first calculate the maximum value of P_{vn}^h for node $n \in \mathcal{N}$, and as long as $P_{vn}^h = 1$ for some $1 \leq h \leq H$ and $v \in V$, it means that node n is in use to host VM(s). After that, we take the sum of $\max_{1 \leq h \leq H, v \in V} P_{vn}^h$ for all the nodes in $\mathcal{N}$ and try to minimize this value. Eq. (8) ensures that each one of k requested VMs must be placed in the network. Eq. (9) ensures that each node does not exceed its storage limit when VMs are placed on it. Eq. (10) makes sure that the specified delay constraint and connection availability of any two VMs under the same placement group are not violated. Eq. (11) ensures that the VM placement availability constraint is obeyed, according to Eq. (5). We note that Eq. (11) can simultaneously calculate the availability of the fully protected placement, partially protected placement, and single placement. For instance, when $H = 2$, Eq. (11) becomes:

$$\prod_{n \in \mathcal{N}} \min_{v \in V}(1 - P_{vn}^1 + P_{vn}^1 A_n) + \prod_{n \in \mathcal{N}} \min_{v \in V}(1 - P_{vn}^2 + P_{vn}^2 A_n) -$$
$$\prod_{n \in \mathcal{N}} \min(\min_{v \in V}(1 - P_{vn}^1 + P_{vn}^1 A_n), \min_{v \in V}(1 - P_{vn}^2 + P_{vn}^2 A_n)) \geq \delta \tag{12}$$

When $P_{vn}^1 = P_{vn}^2$ for all $n \in \mathcal{N}$, Eq. (12) becomes

$$\prod_{n \in \mathcal{N}} \min_{v \in V}(1 - P_{vn}^1 + P_{vn}^1 A_n) \geq \delta$$

which is the VM placement availability constraint for the single placement.

To solve the RVMP problem with SRNG failures, we need to rewrite Eq. (11) in Eq. (13) and keep the objective and all other constraints the same (Eq. (7)-Eq. (10)).

Although inefficient in practice when the problem size is large, the INLP is useful for comparison purposes and demonstrates how accurate the heuristics are. This is shown in Figures 3 and 4 (for smaller size problems).

5.3 Heuristic Algorithm

Algorithm 1: DSR($\mathcal{G}(\mathcal{N}, \mathcal{L}), r(k, c, V, T, A, \delta), H, \alpha$)

```
VP[h][v][n] ← 0 ∀1 ≤ h ≤ H, |v| = k, |n| = N
for h ← 1 to H do
    VP[h] ← DSRPlace(G(N, L), r(k, c, V, T, δ), H, α)
    N ← N\N_x, where N_x denotes a subset of the
      used nodes for already found placement groups.
    if 1 − ∐_{i=1}^{H}(1 − A_{VP[i]}) ≥ δ then
        Call PartiallyDSRPlace(G, VP[h][k][N], r, H)
Return null
```

Algorithm 2: DSRPlace($\mathcal{G}(\mathcal{N}, \mathcal{L}), r(k, c, V, T, A, \delta), H, \alpha$)

```
foreach v_m in V (1 ≤ m ≤ k) do
    v_x ← v_m, Q ← ∅, G^m ← G, P^m[V][N] ← 0
    while Q.Count < k do
        Sort the nodes in G^m by their availabilities in
          the decreasing order n_1, n_2,...,n_N
        Find one node n_a with maximum availability
          to host v_x without violating the delay and
          connection availability constraints with
          already placed VMs, such that s_{n_a} ≥ c_{v_x}
        if Step 5 succeeds then
            P^m[v_x][n_a] ← 1, s_{n_a} ← s_{n_a} − c_{v_x},
              A_{n_a} ← 1, Q.Add(v_x)
        else
            Break;
        χ ← +∞;
        foreach v_i in V\Q do
            foreach v_j in Q do
                if χ > T(v_i,v_j)·α / A(v_i,v_j) then
                    χ ← T(v_i,v_j)·α / A(v_i,v_j), v_x ← v_i
Return P^m with the maximum availability.
```

Our proposed heuristic to solve the RVMP problem, called the Delay-Sensitive and Reliable (DSR) placement algorithm, is shown in Algorithm 1. Instead of placing VMs on nodes, the logic of DSR is to assign nodes to VMs until all the VMs are hosted by the nodes without violating VM delay and connection availability constraints. Since we want to use the least number of nodes to host VMs to satisfy the availability requirement, we gradually increase the amount of finding placement groups. In what follows,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

$$\sum_{h=1}^{H}\prod_{n\in\mathcal{N}}\min_{v\in V}\left(1-P_{vn}^{h}+P_{vn}^{h}A_n\right)\cdot\prod_{1\le i\le g}\min_{v\in V,n\in\mathcal{N}}\left(1-P_{vn}^{h}+P_{vn}^{h}\lambda_i^n\right)-$$
$$\sum_{1\le h<u\le H}\prod_{n\in\mathcal{N}}\min\left(\min_{v\in V}(1-P_{vn}^{h}+P_{vn}^{h}A_n),\min_{v\in V}(1-P_{vn}^{h}+P_{vn}^{h}A_n)\right)\cdot\prod_{1\le i\le g}\min(\min_{v\in V,n\in\mathcal{N}}\left(1-P_{vn}^{u}+P_{vn}^{u}\lambda_i^n\right),\min_{v\in V,n\in\mathcal{N}}\left(1-P_{vn}^{u}+P_{vn}^{u}\lambda_i^n\right))$$
$$+\cdots+(-1)^{H-1}\left(\prod_{n\in\mathcal{N}}\min_{1\le h\le H}(\min_{v\in V}(1-P_{vn}^{h}+P_{vn}^{h}A_n))\right)\cdot\prod_{1\le i\le g}\min_{1\le h\le H}\left(\min_{v\in V,n\in\mathcal{N}}\left(1-P_{vn}^{h}+P_{vn}^{h}\lambda_i^n\right)\right)\ge\delta \quad (13)$$

Algorithm 3: PartiallyDSRPlace($\mathcal{G}, VP[H][k][N], r, H$)

1 Sort the nodes that host VMs in increasing order by nodes' availabilities. Denote this set as $\mathcal{N}_y$.
2 **foreach** $n \in \mathcal{N}_y$ **do**
3 $VB \leftarrow VP$
4 $VB[h][v][n] \leftarrow 0$ for $1 \le h \le H$ and $v \in V$
5 **foreach** *placement group* $h = 1...H$ **do**
6 Try to use its other used nodes to host the VMs that are originally placed by h on n if possible.
7 **if** $1 - \coprod_{i=1}^{H}(1 - A_{VB[i]}) \ge \delta$ **then**
8 $VP \leftarrow VB$, $VB \leftarrow \emptyset$.
9 Return VP

we explain each step of the heuristic algorithm, where the SRNG failures are first not considered.

In Step 1 of Algorithm 1, we first initialize a binary variable $VP[h][v][n]$ representing whether VM $v \in V$ is hosted by node $n \in \mathcal{N}$ under the group h. After that, for placement group h, we call Algorithm 2 to place VMs on nodes in Step 3. The purpose of Step 4 is to avoid different groups to have the same placement result. But this will only happen when a single node's free capacity is far greater than the VM demanding capacity. That is, all the VMs can be placed on the same node and its remaining free capacity is still large enough so that another set of k VMs can be placed on it. In Step 5, we calculate the availability of $VP[1]$, ..., $VP[h]$. If availability value of these h placement groups is no less than δ, we call Algorithm 3 trying to return a partially VM placement solution in order to further reduce the number of used nodes. In Algorithm 3, for each node $n \in \mathcal{N}_y$, where $\mathcal{N}_y$ stores the nodes in the increasing order by their availabilities, we first clear all the VMs resident on n. For each placement group p, we try to use its other used nodes to host the VMs that are originally placed by it on n. For simplicity, we apply a greedy approach: for each one (say n_u) of used nodes by placement group h, we let n_u host the VMs which are originally placed on n by p as many as possible. After that, we calculate whether the whole availability still satisfies δ. If so, we assign this partially placement solution to VP. Next, we will explain the details of Algorithm 2, which is to find a single placement.

In Step 1 of Algorithm 2 we start with each $v_m \in V$, and assign it to v_x in Step 2. We use a queue Q to store the VMs already placed, and initially it is set to empty. Besides, we also define variable $P^m[v][n]$ to indicate whether VM $v \in V$ is hosted by node $n \in \mathcal{N}$ corresponding to the placement group starting with VM v_m. As long as Q's count is less than k, Step 4-Step 9 are going to assign nodes to host unassigned VMs. Step 5 tries to find a node n_a with maximum availability whose capacity should be at least $c(v_x)$. Moreover, if v_x is placed on n_a, it should not violate the delay and connection availability constraints with already hosted VMs. If it succeeds, in Step 7, the capacity of n_a is reduced by c_{v_x}, the availability of n_a is changed to 1, and v_x is added to Q. The reason to change a node's availability is that if some nodes have been used to host the existing VM(s), then the availability for these nodes to host other (unassigned) VMs is 1. So we need to change its "availability" after each iteration of covering VM(s). If such a node cannot be found in Step 5, this indicates that not all the VMs are covered and we consider this placement group should "jointly" place uncovered VMs with one of $H-1$ placement groups found in Algorithm 1. The algorithm then breaks in Step 9. Following that, Step 10-Step 14 search for an unsigned VM, which has the smallest value of $\frac{T(v_i,v_j)\cdot\alpha}{A(v_i,v_j)}$ to the VMs already placed, where α is a user given value. By doing this, we want to find an unsigned VM that has "smaller" path delay and "greater" connection availability constraints with already placed VMs, and assigns it to v_x. The motivation here is that we always first place the VM which has a more critical requirement in terms of path delay and connection availability. When Q's count is equal to k, it indicates that all the VMs have been hosted, which means we get a "complete" placement group. Finally, in Step 15, the algorithm returns a placement group with the biggest availability from k already determined single placements.

To solve the RVMP problem with SRNG failures, Alg. 1-3 remain the same except:

- In Step 4 of Algorithm 2 and Step 1 of Algorithm 3, we sort the nodes in $\mathcal{G}^m$ by the product of their availabilities and non-occurring probabilities of all their belonging SRNG events in a decreasing order and an increasing order, respectively.
- In Step 5 of Algorithm 2, we will find one node n_a with maximum node availability multiplied by the non-occurring probabilities of SRNG events set $R_{n_a} \backslash R_x$, where R_x denotes the set of SRNG events that n_a belongs to but has already been considered/counted by the other nodes from Step 4 to Step 14. The reason is that one unique SRNG event can only occur once, so we cannot calculate its value under the same placement group more than once.

The time complexity of Algorithm 2 can be calculated like this: There are k VMs in total in Step 1, and Step 3 has also k iterations. Sorting algorithm for instance like insertion sort in Step 4 takes $O(N\log(N))$ time, and Step 5 has a complexity of $O(N)$. Step 9-Step 13 consume at most $O(k^2)$ time. Therefore, the whole complexity of Algorithm 2 is $O(k^2(N\log(N)+k^2))$. In Algorithm 3, Step

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

1 consumes $O(N\log(N))$ time via insertion sort and Steps 2-8 consume $O(N^2H)$ time, leading to a whole complexity of $O(N(\log N + NH))$. Consequently, the whole time complexity of Algorithm 1 is $O(k^2H(N\log(N) + k^2))$, since it calls at most H times of Algorithm 2.

5.4 Simulations

The simulations are run on a desktop PC with 2.7 GHz and 8 GB memory. We use an Intel(R)Core(TM)i5-4310M CPU 2.70GHz x64-based processor in our simulations. We use IBM ILOG CPLEX 12.6 to implement the proposed INLP. All the heuristics are implemented by C# and compiled on Visual Studio 2015 (using .NET Framework 4.5).

We set $\alpha = 1$ for our heuristic DSR. We compare our exact INLP and heuristic DSR with two heuristics, namely (1) Greedy Placement (GP) and (2) Random Placement (RP). These 2 algorithms follow the similar routine with Algorithm 1, except: (1) in Step 6, they directly return the placement result if its availability is satisfied, instead of checking partially placement solution, and (2) they call different heuristics in Step 3 (different from Algorithm 2), which we specify as follows:

- GP (or RP): It first selects a node with greatest availability (or randomly selects a node) and places as many VMs as possible on it under its storage limit. It then selects the second largest availability node (or randomly selects the second node) and places as many of the remaining VMs as possible, which should also satisfy the delay and connection availability constraints with the VMs already placed. This procedure continues until all the VMs are placed or all the nodes have been iterated.

In the following, we first test the algorithms for the RVMP problem without SRNG failures for both 16-node and 100-node networks, and then evaluate them for the RVMP problem with SRNG failures for a 100-node network.

5.4.1 16-node network without SRNG failures

We first conduct simulations on a 16-node network. If we set c (VM demanding capacity) relatively too small, then by placing as many VMs as possible on one node it may return a solution. If we set c relatively too big, then the solution may not exist. Therefore, we let the node's capacity be at most three times of the requested bandwidth of one single VM, by which we want to challenge the algorithm to find the solution. Consequently, the simulation parameters are set like this: the node capacities are randomly distributed between 100 and 200 units, and the node availabilities are randomly distributed among the set $\{0.99, 0.999, 0.9995, 0.9999\}$. For each request $r(k, c, V, T, A, \delta)$, $k \in [3, 5]$, $c \in [60, 130]$, each element in the delay matrix T is between 15 and 25, each element in the connection availability matrix A is among the set $\{0.999, 0.9999\}$, and δ is in the set $\{0.999, 0.9999, 0.99999, 0.999999\}$. We randomly generate 100 requests for $k = 3, 4, 5$, respectively. With respect to $F(m, n, \eta, D)$, when $\eta = 0.999$ for link (m, n), it returns 1 when the delay is at most D, where D is randomly chosen between $[10, 20]$, otherwise it returns 0; when $\eta = 0.9999$ for link (m, n), it returns 1 when the delay is at most D, where D is randomly chosen between $[20, 30]$, otherwise it returns 0. We set $H = 2$ and 3.

We first evaluate the performance of the algorithms in terms of Acceptance Ratio (AR), which is defined as the number of accepted requests over all the requests (between 0 and 1). Figs. 3(a) and 4(a) show that the exact INLP always achieves the highest AR. DSR has a close performance with INLP, and it outperforms the other two heuristics. Besides, we notice that for the same algorithm, it achieves higher AR value when H increases, since more VM replicas are allowed to be placed for a higher H.

Next, we compare the algorithms in terms of Average Number of Used Nodes (ANUN). The ANUN is defined as the total number of nodes consumed by all the accepted requests divided by the number of accepted requests. From Figs. 3(b) and 4(b), we see that the achieved ANUN value by RP when $k = 4, 5, H = 2$, and when $k = 5, H = 3$ is the (or second) lowest. This is because its acceptance ratio in those scenarios is too low (under 15%), and it only finds solutions for some "easier" requests. Except for those cases, the INLP achieves the minimum value of ANUN, and our proposed DSR obtains the second lowest ANUN value. RP obtains a lower ANUN value than GP when $k = 4$ and $H = 3$, since it is regarded to place more shared VMs. From above, we observe that even under the constrained simulation setup, the exact INLP can always accept most requests and consume the least amount of nodes as well, which validates its correctness.

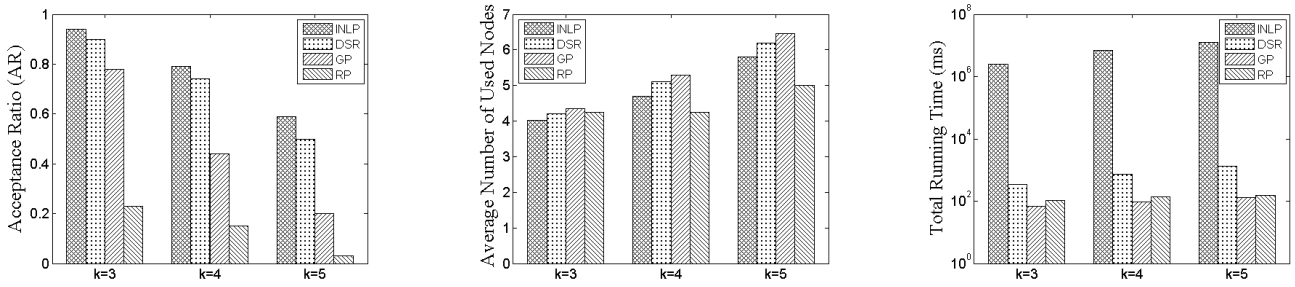
Finally, Figs. 3(c) and 4(c) present the total running time over 100 requests (in log scale). The INLP is significantly more time-consuming than all the 3 heuristics. The DSR, on the other hand, has a slightly higher running time than the other two heuristics, but it pays off by having a higher AR as shown in Figs. 3(a) and 4(a), and lower ANUN shown in Figs. 3(b) and 4(b).

5.4.2 100-node network without SRNG failures

In this subsection, we simulate a 100-node complete graph, where the node capacities are randomly distributed in $[1000, 2000]$. The other simulation setup follows the same with Section 5.4.1. Since the problem sizes increase largely, the INLP becomes very time-consuming and it keeps running for at least one day without returning a feasible solution. We therefore only evaluate the heuristic algorithms. Due to the lack of the INLP, we generate 100 sets of 100 traffic requests for each $k = 100, 150, 200$ requested VMs, respectively, and evaluate all the heuristic algorithms for those 100 sets of 100 traffic requests (100 runs). By doing this, we want to establish confidence on the performance of heuristics. Figs. 5 and 6 depict the AR, ANUN and running time (in log scale) of all these algorithms, where the confidence interval is set to 95%. The 95% confidence interval is calculated for all the figures, but in those where it is not visible, the interval is negligibly small[4]. Similar to Section 5.4.1, DSR always achieves better performance than the other 2 heuristics in terms of AR (see Figs. 5(a) and 6(a)) and ANUN (see Figs. 5(b) and 6(b)). On the other hand, Figs. 5(c)

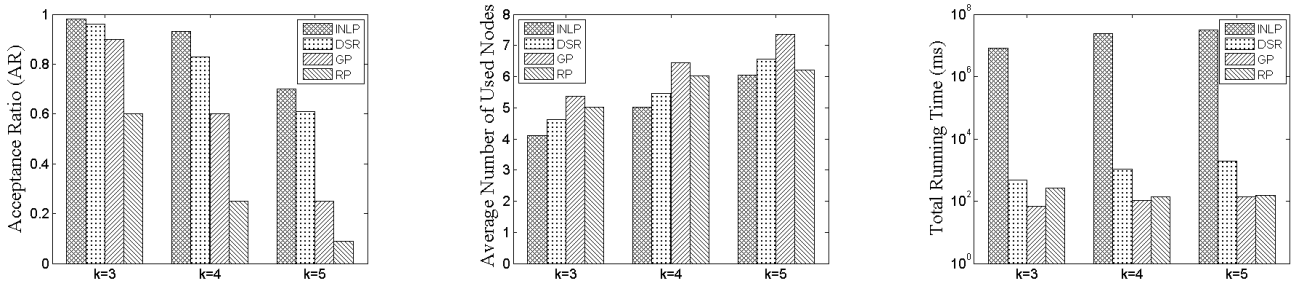
4. We note here that some plots are log-scale that additionally contributes to the confidence interval visibility.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems



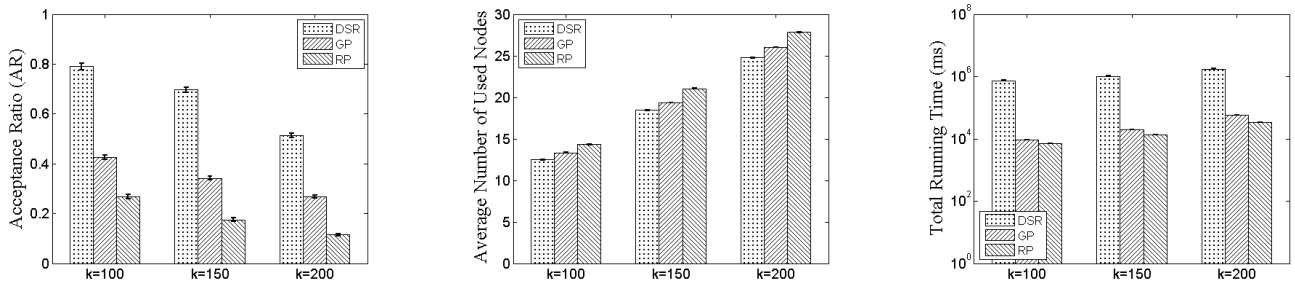
(a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (c) Running Time

Fig. 3: Simulation results over 100 requests when $H = 2$ for 16-node network: (a) Acceptance Ratio (AR), (b) Average Number of Used Nodes (ANUN) and (c) Running Time.



(a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (c) Running Time

Fig. 4: Simulation results over 100 requests when $H = 3$ for 16-node network: (a) Acceptance Ratio (AR), (b) Average Number of Used Nodes (ANUN) and (c) Running Time.



(a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (c) Running Time

Fig. 5: Simulation results over 100 sets of 100 requests (95% confidence interval) when $H = 2$ for 100-node network: (a) Acceptance Ratio (AR), (b) Average Number of Used Nodes (ANUN) and (c) Running Time.

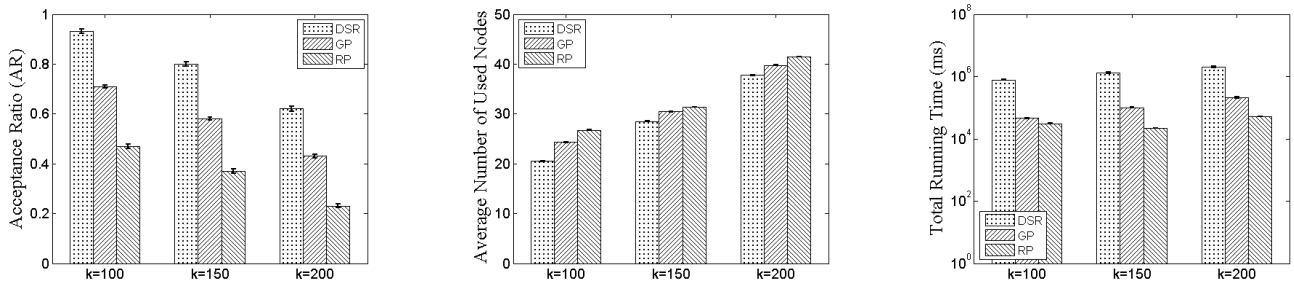
and 6(c) show that DSR is more time consuming than the other heuristics, but it is still acceptable since it acquires higher AR and lower ANUN values. Another observation is that in this larger network scenario, RP obtains the highest ANUN value. This reveals that RP performs more poorly because of its randomness when the problem size grows.

5.4.3 100*-node network with SRNG failures*

It is assumed that there are in total 15 SRNG events, and each SRNG event occurs with the probability in the set $\{0.000001, 0.000002, 0.000003, 0.000004, 0.000005\}$. Each server node is associated with at most 5 SRNG events. The other simulation setup follows the same with Section 5.4.2 and we also evaluate all the three algorithms by 100 runs to establish confidence. Since more SRNG events are induced for each node, the total VM placement availability for the same set of nodes will decrease according to Eq. (6), causing the optimal solution not to exist for when $\delta > 0.9999$. Due to space limits, we only present the results for $H = 2$ in Fig. 7, where a confidence interval is set to 95%. Similar to Sections 5.4.1 and 5.4.2, DSR can obtain a better performance than the other 2 heuristics in terms of AR (see Fig. 7(a)) and ANUN (see Fig. 7(b)), but this comes at the expense of a higher running time as shown in Fig. 7(c) (in log scale). Due to the reason for incurring SRNG events, for all three algorithms, we can see that the achieved AR value in Fig. 7(a) is lower than Fig. 5(a), and the obtained ANUN value in Fig. 7(b) is higher than Fig. 5(b).

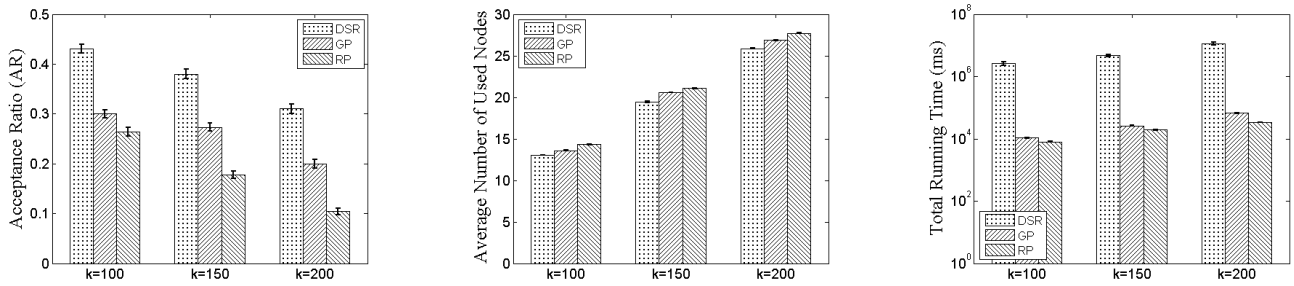
In all, we conclude that the exact INLP can be used as an optimal solution when the computation speed is not a big

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems



(a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (c) Running Time

Fig. 6: Simulation results over 100 sets of 100 requests (95% confidence interval) when $H = 3$ for 100-node network: (a) Acceptance Ratio (AR), (b) Average Number of Used Nodes (ANUN) and (c) Running Time.



(a) Acceptance Ratio (AR) (b) Average Number of Used Nodes (c) Running Time

Fig. 7: Simulation results over 100 sets of 100 requests (95% confidence interval) when $H = 2$ for 100-node network with SRNG failures: (a) Acceptance Ratio (AR), (b) Average Number of Used Nodes (ANUN) and (c) Running Time.

concern. However, as the problem size increases, its running time will increase exponentially. On the contrary, our proposed DSR is a good compromise between performance and running time, and it is the preferred choice for when the VM placement request needs to be computed on-the-fly.

6 Availability-Based Delay-Constrained Routing Problem

In the RVMP problem, we do not consider the link delay and availability, and assume that the function $F(m, n, \eta, D)$ is given. In this section, we study how to find a connection over at most w link-disjoint paths between a node pair, such that the connection availability is no less than η and each path delay is no more than D. For completeness, let us first formulate the connection availability calculation, which is introduced in [18], [19].

6.1 Connection Availability and Problem Definition

Similar to the node availability, we assume that the link availability is equal to the product of availabilities of all its components (e.g., amplifiers). If a path p contains the links $l_1, l_2, l_3, \ldots, l_m$, and their corresponding (independent) availabilities are denoted by $A_{l_1}, A_{l_2}, A_{l_3}, \ldots, A_{l_m}$, then the availability of this (unprotected) path (represented by A_ψ) is equal to $A_\psi = A_{l_1} \cdot A_{l_2} \cdot A_{l_3} \cdots \cdot A_{l_m}$. If we take the $-\log$ of the link availabilities, finding a path with the highest availability is equivalent to the shortest path problem [27].

When, for a single connection (i.e., a single path), there are $w \geq 2$ paths $\psi_1, \psi_2, \ldots, \psi_w$ with availabilities represented by $A_{\psi_1}, A_{\psi_2}, \ldots, A_{\psi_w}$, the connection availability indicates the probability that at least one path is operational. We consider two cases, namely: (1) fully link-disjoint paths: these w paths have no links in common, and (2) partially link-disjoint paths: at least two of these w paths traverse a common link. In case (1), the availability (represented by A^w_{FL}) can be calculated as follows:

$$A^w_{FL} = 1 - \prod_{i=1}^{w} (1 - A_{\psi_i}) \tag{14}$$

If we use Eq. (14) to calculate the availability for the partially link-disjoint case, the probability that the overlapping links operate (or the availability of the overlapping links) will be counted more than once. To solve this, we can analogously apply the operators $\circ$ and $\coprod$ introduced in Section 3. Assuming there are w partially link-disjoint paths $\psi_1, \psi_2, \ldots, \psi_w$, the availability (represented by A^w_{PL}) of w partially link-disjoint paths can be calculated as:

$$A^w_{PL} = 1 - \coprod_{i=1}^{w} (1 - A_{\psi_i}) \tag{15}$$

Let us use an example to explain how to calculate the connection availability for fully and partially link-disjoint paths, where w is set to 2 for simplicity. In Fig. 8 where the link availability is labeled on each link, paths $s - a - t$ and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

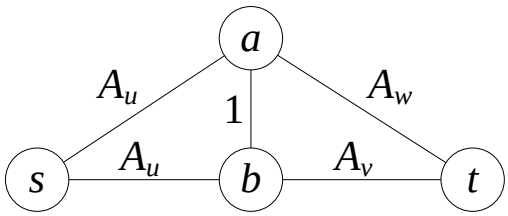


Fig. 8: Availability calculation of a pair of fully and partially link-disjoint paths.

$s-b-t$ are fully link disjoint. According to Eq. (14), their availability is equal to:

$$\begin{aligned}&1-(1-A_u\cdot A_w)\cdot(1-A_u\cdot A_v)\\=&1-(1-A_u\cdot A_v-A_u\cdot A_w+\underline{A_u}\cdot A_w\cdot\underline{A_u}\cdot A_v)\\=&A_u\cdot A_v+A_u\cdot A_w-\underline{A_u^2}\cdot A_w\cdot A_v\end{aligned}\tag{16}$$

On the other hand, paths $s-a-t$ and $s-a-b-t$ are two partially link-disjoint paths. According to Eq. (15), the connection availability can be calculated as follows:

$$\begin{aligned}&1-(1-A_u\circ A_w)\circ(1-A_u\circ A_v)\\=&1-(1-A_u\circ A_v-A_u\circ A_w+\underline{A_u}\circ A_w\circ\underline{A_u}\circ A_v)\\=&A_u\cdot A_v+A_u\cdot A_w-\underline{A_u}\cdot A_w\cdot A_v\end{aligned}\tag{17}$$

Next, we formally define the Availability-Based Delay-Constrained Routing (ABDCR) problem as follows:

Definition 2. Given is a network represented by $G(\mathcal{N},\mathcal{L})$, where $\mathcal{N}$ represents the set of N nodes and $\mathcal{L}$ denotes the set of L links. Each link $l\in\mathcal{L}$ is associated with an availability value A_l and a delay value d_l. For a communication request represented by $r(s,t,\eta,D)$, where s and t denote the source and destination, η $(0<\eta\leq 1)$ represents the connection availability requirement and D indicates the delay constraint, the Availability-Based Delay-Constrained Routing (ABDCR) problem is to establish a connection over at most w (partially) link-disjoint paths, such that the connection availability is at least η and each path has a delay no more than D.

In the ABDCR problem, we regard that each request corresponds to the communication between each VM pair which is resident on different nodes. When, the delay constraint is not imposed on each path, the ABDCR problem is equivalent to the Availability-Based Path Selection (ABPS) problem [18], [19]. In [18], [19] we have proved that the ABPS problem is NP-hard for $w\geq 2$ and cannot be approximated to an arbitrary degree. Therefore, the ABDCR problem for $w\geq 2$ is also NP-hard and cannot be approximated to an arbitrary degree. When $w=1$, the ABDCR problem is equivalent to the Multi-Constrained Routing problem, which is also NP-hard [28]. In the following, we propose an exact algorithm and two heuristics to solve the ABDCR problem.

6.2 Exact Algorithm

To solve the ABDCR problem exactly, we apply a modified Dijkstra's algorithm by letting each node store as many subpaths as possible, which is similar to the exact algorithm for solving the multi-constrained routing problem [28]. We start with some notations used in the algorithm:

$sus[u][m]$: the parent node of node u for its stored m-th subpath from s to u.

$avb[u][m]$: the availability value stored at node u for its stored m-th subpath from s to u.

$delay[u][m]$: the delay value stored at node u for its stored m-th subpath from s to u.

$counter[u]$: the number of stored subpaths of node u.

$sp[u][m]$: node u's stored m-th subpath from s to u.

$adj(u)$: the set of adjacent nodes of node u.

The pseudo code of the exact algorithm for solving the ABDCR problem when $w=1$ is given in Algorithm 4.

Algorithm 4: ABDCRw1(G,s,t,η,D)

```
1  Q ← s, avb[s][1] ← 1, delay[s][1] ← 0,
     avb[i][m] ← +∞, sus[i][m] ← i, counter[s] ← 1,
     counter[i] ← 0, ∀i ∈ N\{s}
2  while Q ≠ ∅ do
3      u[m] ← Extract-min(Q)
4      if u == t && avb[u][m] ≥ η then
5          Return the path sp[u][m]
6      else
7          foreach v ∈ adj(u) do
8              if delay[u][m] + d_uv ≤ D then
9                  counter[v] = counter[v] + 1
10                 Assign the availability of subpath
                     sp[u][m]-(u,v) to avb[v][counter(v)] ;
11                 delay[v][counter(v)] ← delay[u][m] + d_uv
12                 sus[v][counter(v)] ← u
13                 Insert (Q, v, counter(v))
```

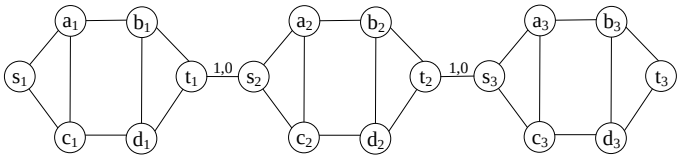


Fig. 9: An example of graph transformation for solving the ABDCR problem when $H=3$.

When $w>1$ in the ABDCR problem, we could first duplicate the originate graph $\mathcal{G}$ into w copies $\mathcal{G}^1$, $\mathcal{G}^2$,..., $\mathcal{G}^w$. After that, we create link (t_i,s_j) to connect each graph copy with availability 1 and delay 0 for $1\leq i\leq w$, $j=i+1$ except for $i=w$, where $n\in\mathcal{G}$ and $n_i\in\mathcal{G}^i$. By doing this, we obtain an auxiliary graph $\mathcal{G}^l$, where the source node is s_1 and the destination node(s) is t_i. In this context, if we find a path ψ from s_1 to t_i in $\mathcal{G}^l$, we can map ψ into the original graph $\mathcal{G}$, where we can get i (link-disjoint) paths from s to t. For example, Fig. 9 depicts the resulted auxiliary graph after transformation from the topology shown in Fig. 2 when $H=3$. Suppose we find a path ψ_1: s_1-a_1-b_1-t_1-s_2-c_2-d_2-t_2, then it is equivalent to say that we find two link-disjoint paths s-a-b-t and s-c-d-t. On the other hand, if there is a path ψ_2: s_1-a_1-b_1-t_1-s_2-a_2-b_2-t_2, then it can be mapped to the same path s-a-b-t in the original graph. In this context,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

we regard that the availability of ψ_2 in $\mathcal{G}^l$ is the same with s-a-b-t in $\mathcal{G}$.

As a result, we could slightly modify Algorithm 4 to solve the ABDCR problem for $w > 1$ as follows ($1 \leq i \leq w$):

- In Step 4, the condition should only be **if** $u == t_i$ && the availability of i link-disjoint paths (after mapping them to the original graph) is greater than or equal to η.
- In Step 10, for each node v_i, its avb value is calculated based on i link-disjoint paths after mapped to $\mathcal{G}$ together with the subpath from s_i to v_i, according to Eq. (5). For example, suppose in Step 3 node a_2 is extracted and its stored subpath is s_1-a_1-b_1-t_1-s_2-a_2. Now in Step 7, suppose the neighbor node c_2 is selected to update its avb value according to node a_2. Therefore the subpath from s_1 to c_2 is s_1-a_1-b_1-t_1-s_2-a_2-c_2, which are paths s-a-b-t and s-a-c after mapped to the original graph.
- In step 11, s_i does not update its $delay$ value. It also sets $delay[s_i][counter(s_i)] = 0$.

It is worthwhile to mention that our proposed exact algorithm can also solve the ABDCR problem in Shared-Risk Link Group (SRLG) networks. Similar to SRNG, the links in the same SRLG will fail simultaneously if the group they belong to fails. For example, in optical networks [29], several fibers may reside in the same duct and a cut of the duct would cut all fibers in it. One duct in this context corresponds to one distinct SRLG. To solve it, we only need to change the connection availability calculation in Step 4 of Algorithm 4. More details of the connection availability in SRLG networks can be found in [18], [19].

The time complexity of algorithm 4 can be computed as follows. Let $Z_{\max}$ denote the maximum number of subpaths for each node to store, then in Step 2, Q contains at most $Z_{\max}N$ subpaths. According to [30], $Z_{\max} \leq \lfloor e(N-2)! \rfloor$, where $e \approx 2.718$ is the Euler's number. When using a Fibonacci heap to structure the heap, selecting the minimum cost path has a time complexity of $O(\log(Z_{\max}N))$ [31] in Step 3. Step 7-Step 13 take at most $O(Z_{\max})$ time for each link to be iterated thus resulting in $O(Z_{\max}L)$ time; because for a fixed link, the steps within the inner loop (Steps 8-13) all cost $O(1)$ time. Hence, the overall time complexity of Algorithm 4 is $O(Z_{\max}N\log(Z_{\max}N) + Z_{\max}L)$. Similarly, when $w > 1$ for the exact algorithm, the overall time complexity is $O(Z_{\max}wN\log(Z_{\max}wN) + Z_{\max}wL)$.

6.3 Heuristic algorithms

We propose two heuristic algorithms to solve the NP-hard ABDCR problem. The first one is called SeqTAMCRA: it leverages on TAMCRA [28], which is a heuristic to solve the multi-constrained routing problem. The procedure of SeqTAMCRA is the following: it iteratively runs TAMCRA, so in each iteration, we may obtain a path with the biggest availability and delay no more than D. After each iteration, the traversed links will be pruned. This procedure continues until the connection availability is satisfied or the number of paths is bigger than w.

The second heuristic is called TADRA, Tunable Availability-based Delay-constrained Routing Algorithm, and it is identical to the exact algorithm except that the number (variable $counter$) of stored paths for each node should not exceed a given value (say M). For instance, Step 8 of Algorithm 4 should be rewritten as:
if $counter[v] \leq M$ && $delay[u][m] + d_{uv} \leq D$ **then**.

6.4 Simulations

In order to verify the proposed algorithms, we conduct simulations on two networks[5]: USANet, displayed in Fig. 10, which is a realistic carrier backbone network, and GÉANT, shown in Fig. 11, which is a pan-European communications infrastructure. The link availability values are from the set $\{0.99, 0.999, 0.9999\}$, and the link delays are set between 10 and 25. We randomly generate 1000 requests, and for each request $r(s, t, \eta, D)$, s and t are randomly generated, η is among the set $\{0.9995, 0.9996, 0.9997, 0.9998, 0.9999\}$, and $D \in [15, 25]$. We set $w = 1, 2, 3$. For both SeqTAMCRA and TADRA, the maximum number of stored paths is set to wN, where w is the number of maximum link-disjoint paths and N is the number of nodes in the network.

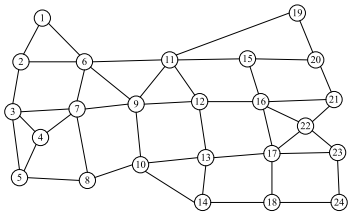


Fig. 10: USA carrier backbone network.

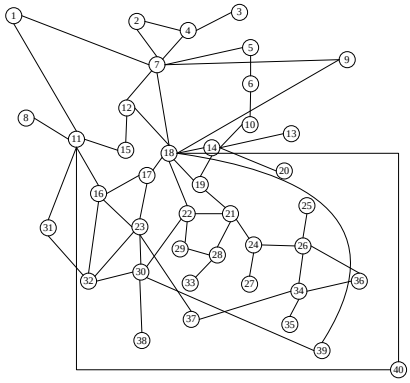
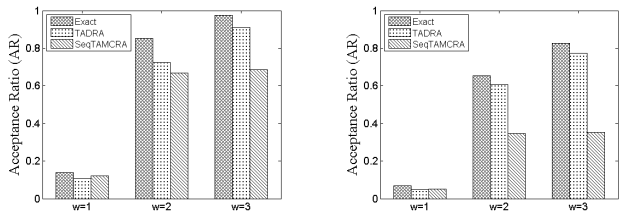


Fig. 11: GÉANT pan-European research network.

Figs. 12(a) and 12(b) depict the Acceptance Ratio (AR) of the 3 algorithms, which is defined by the number of accepted requests divided by the total number of requests. The exact algorithm can always achieve the highest AR value, which also verifies its correctness. TADRA obtains close to optimal performance when $w = 2$ and $w = 3$, while SeqTAMCRA performs well when $w = 1$. The reason is that when $w = 1$ SeqTAMCRA dynamically maintains wN best paths for each node compared to the TADRA, so it can achieve a better performance. When $w > 1$, after finding a

5. Since most of typical data center network topologies are tree-like (e.g., Fat-Tree, BCube), the number of link-disjoint paths between node pairs is limited. Hence, to examine the algorithms more thoroughly, we choose well-connected backbone networks for the evaluation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2693273, IEEE Transactions on Parallel and Distributed Systems

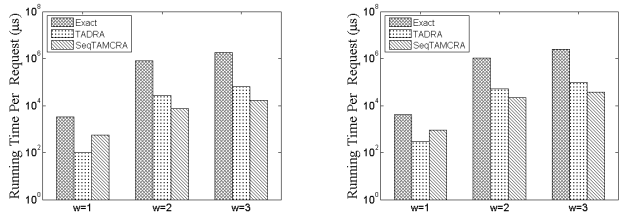


(a) USANet (b) GÉANT

Fig. 12: Acceptance ratio in two networks: (a) USANet (b) GÉANT .

feasible path by SeqTAMCRA, pruning the used links will prevent it to find link-disjoint paths in some cases, leading to worse performance. Moreover, dynamically maintaining fixed number of best paths only works for when $w = 1$, since the connection availability calculation is non-linear when $w \geq 2$. Therefore, the exact algorithm and TADRA cannot adopt this technique to improve the efficiency for finding feasible paths. Nevertheless, we could jointly use TADRA when $w > 1$ and SeqTAMCRA when $w = 1$ as a heuristic combination.

When the solution does not exist, the exact algorithm needs much longer time to finish, we therefore only show the algorithms' running times (in log scale) for all their accepted requests in Fig. 13. Even in this case, we see that the exact algorithm is still more time consuming than the others. TADRA requires more running time than SeqTAMCRA when $w > 1$, since its graph (input) size increases w times.



(a) USANet (b) GÉANT

Fig. 13: Running time per request in two networks: (a) USANet (b) GÉANT .

7 Conclusion

In this paper, we have first studied the Reliable VM Placement (RVMP) problem. We have shown that the RVMP problem is NP-hard, and cannot be approximated to an arbitrary degree. To solve it, we have proposed an exact INLP as well as an efficient heuristic, and compare these algorithms with another two heuristic modifications. The simulation results reveal that, our proposed heuristic can always achieve a better performance in terms of acceptance ratio and the number of used nodes than the modified heuristics, although it consumes an (acceptably) higher running time. On the other hand, the exact INLP can always achieve the best performance, but its running time is significantly larger than all the heuristics. Following that, we have studied the Availability-Based Delay-Constrained Routing (ABDCR) problem. We have shown that the ABDCR problem is NP-hard and have proposed both an exact algorithm and two heuristics to solve it. Finally, we have tested these 3 algorithms via simulations on two networks. The simulation results indicate that the exact algorithm can always achieve the highest acceptance ratio, but this comes at the expense of much higher running time. Meanwhile, SeqTAMCRA and TADRA return close to optimal result in a shorter time for $w = 1$ and $w > 1$, respectively, which suggests a combination of use when the computation time is a big concern.

In reality, the cloud provider can first solve the ABDCR problem via the proposed algorithms in Section 6.2 and/or 6.3 for different node pairs in the network. Those returned solutions serve as the input for the RVMP problem. At last, the cloud provider can solve the RVMP problem by using the proposed solutions in Section 5.2 and/or 5.3. This is one possible scenario how a practitioner can face both problems in Sections 5 and 6, and can apply our proposed solutions in the sequential order. In case the network (problem) size is too large but the computation time needs to be short, one possible approach is to first exclude some "poor availability" nodes from the graph and then run the algorithm(s) on the remainder of the network.

Acknowledgments

This research was funded by the joint EU FP7 Marie Curie Actions CleanSky Project, Contract No. 607584.

References

[1] S. Yang, P. Wieder, and R. Yahyapour, "Reliable virtual machine placement in distributed clouds," in *Proc. of 8th IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2016, pp. 1–7.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," *Communications of the ACM*, vol. 53, no. 6, p. 50, 2010.

[3] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014.

[4] Z. Á. Mann, "Allocation of virtual machines in cloud data centers—-a survey of problem models and optimization algorithms," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 11, 2015.

[5] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. of IEEE INFOCOM*, 2012, pp. 963–971.

[6] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware vm placement for cloud systems," in *Proc. of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2012, pp. 498–506.

[7] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. of IEEE INFOCOM*, 2012, pp. 2876–2880.

[8] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[9] A. Israel and D. Raz, "Cost aware fault recovery in clouds," in *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013, pp. 9–17.

[10] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Proc. of 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 700–709.

[11] Y. Zhu, Y. Liang, Q. Zhang, X. Wang, P. Palacharla, and M. Sekiya, "Reliable resource allocation for optically interconnected distributed clouds," in *Proc. of IEEE International Conference on Communications (ICC)*, 2014, pp. 3301–3306.

[12] X. Li and C. Qian, "Traffic and failure aware VM placement for multi-tenant cloud computing," in *IEEE 23rd International Symposium on Quality of Service (IWQoS)*, 2015, pp. 41–50.

[13] Z. Yang, L. Liu, C. Qiao, S. Das, R. Ramesh, and A. Y. Du, "Availability-aware energy-efficient virtual machine placement," in *IEEE International Conference on Communications (ICC)*, 2015, pp. 5853–5858.

[14] L. Song, J. Zhang, and B. Mukherjee, "Dynamic provisioning with availability guarantee for differentiated services in survivable mesh networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 3, pp. 35–43, 2007.

[15] Q. She, X. Huang, and J. Jue, "How reliable can two-path protection be?" *IEEE/ACM Transactions on Networking*, vol. 18, no. 3, pp. 922–933, 2010.

[16] H. Luo, L. Li, and H. Yu, "Routing connections with differentiated reliability requirements in WDM mesh networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 253–266, 2009.

[17] H.-W. Lee, E. Modiano, and K. Lee, "Diverse routing in networks with probabilistic failures," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, pp. 1895–1907, 2010.

[18] S. Yang, S. Trajanovski, and F. Kuipers, "Availability-based path selection," in *Proc. of 6th IEEE International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2014, pp. 39–46.

[19] S. Yang, S. Trajanovski, and F. A. Kuipers, "Availability-based path selection and network vulnerability assessment," *Networks*, vol. 66, no. 4, pp. 306–319, 2015.

[20] J. I. McCool, *Probability and Statistics With Reliability, Queuing and Computer Science Applications*. Taylor & Francis, 2003.

[21] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. of the 1st ACM symposium on Cloud computing*, 2010, pp. 193–204.

[22] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, 2011, pp. 350–361.

[23] P. Datta and A. K. Somani, "Graph transformation approaches for diverse routing in shared risk resource group (srrg) failures," *Computer Networks*, vol. 52, no. 12, pp. 2381–2394, 2008.

[24] C. Kachris, K. Bergman, and I. Tomkos, *Optical interconnects for future data center networks*. Springer Science & Business Media, 2012.

[25] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., 1979.

[26] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *IEEE INFOCOM*, 2013, pp. 647–655.

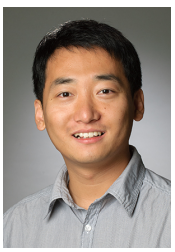
[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[28] P. Van Mieghem and F. A. Kuipers, "Concepts of exact QoS routing algorithms," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 851–864, 2004.

[29] B. Mukherjee, *Optical WDM Networks*. Springer Science & Business Media, 2006.

[30] P. Van Mieghem, "Paths in the simple random graph and the waxman graph," *Probability in the Engineering and Informational Sciences*, vol. 15, no. 04, pp. 535–555, 2001.

[31] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

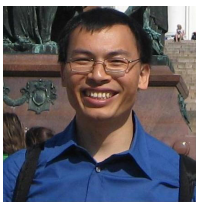


Song Yang received the B.S. degree in software engineering and the M.S. degree in computer science from Dalian University of Technology, Dalian, Liaoning, China, in 2008 and 2010, respectively, and the Ph.D. degree from Delft University of Technology, The Netherlands, in 2015. He is currently a postdoc researcher in GWDG. His research interests focus on network optimization algorithms in optical networks, stochastic networks and data center networks.

Philipp Wieder is deputy leader of data center of the GWDG at the University of Göttingen, Germany. He received his doctoral degree from TU Dortmund in Germany. He is active in the research areas on clouds, grid and service oriented infrastructures for several years. His research interest lies in distributed system, service level agreements and resource scheduling. He has been actively involved in the FP7 IP PaaSage, SLA@SOI and SLA4D-Grid projects.

Ramin Yahyapour is full professor at the Georg-August University of Göttingen. He is also managing director of the GWDG, a joint compute and IT competence center of the university and the Max Planck Society. Dr. Yahyapour holds a doctoral degree in Electrical Engineering and his research interest lies in the area of efficient resource management in its application to service-oriented infrastructures, clouds, and data management. He is especially interested in data and computing services for eScience. He gives lectures on parallel processing systems, service computing, distributed systems, cloud computing, and grid technologies. He was and is active in several national and international research projects. Ramin Yahyapour serves regularly as reviewer for funding agencies and consultant for IT organizations. He is organizer and program committee member of conferences and workshops as well as reviewer for journals.

Stojan Trajanovski is a research and data scientist in Philips Research and a visiting researcher at Delft University of Technology in The Netherlands. He was a postdoctoral researcher at the University of Amsterdam. He received his Ph.D. degree (*cum laude*, 2014) from Delft University of Technology and his master degree in Advanced Computer Science (*with distinction*, 2011) from the University of Cambridge, United Kingdom. He also holds an MSc degree in Software Engineering (2010) and a Dipl. Engineering degree (*summa cum laude*, 2008) from Ss. Cyril and Methodius University in Skopje. He successfully participated at international science olympiads, winning a bronze medal at the International Mathematical Olympiad (IMO) in 2003. His main research interests include network science, complex networks, network robustness, game theory, optimization algorithms and machine learning.



Xiaoming Fu received his Ph.D. in computer science from Tsinghua University, Beijing, China in 2000. He was then a research staff at the Technical University Berlin until joining the University of Göttingen, Germany in 2002, where he has been a professor in computer science and heading the Computer Networks Group since 2007. He has spent research visits at universities of Cambridge, Uppsala, UPMC, Columbia, UCLA, Tsinghua, Nanjing, Fudan, and PolyU of Hong Kong. Prof. Fu's research interests include network architectures, protocols, and applications. He is currently an editorial board member of IEEE Communications Magazine, IEEE Transactions on Network and Service Management, and Elsevier Computer Communications, and has served on the organization or program committees of leading conferences such as INFOCOM, ICNP, ICDCS, MOBICOM, MOBIHOC, CoNEXT, ICN and COSN. He is an IEEE Senior Member and an IEEE Communications Society Distinguished Lecturer.